Virtual Machines Performance Modeling with Support Vector Regressions

Shing H. Doong
ShuTe University
Taiwan 824
tungsh@stu.edu.tw

Chih C. Lai
National University of Kaohsiung,
Taiwan 811
cclai@nuk.edu.tw

Shie J. Lee
National Sun Yat-Sen University,
Taiwan 804
leesj@mail.ee.nsysu.edu.tw

Chen S. Ouyang
I-Shou University
Taiwan 840
ouyangcs@isu.edu.tw

Chih H. Wu
National University of Kaohsiung,
Taiwan 811
johnw@nuk.edu.tw

***Abstract*—Virtualization is a key technology in cloud computing to render on-demand provisioning of virtual services. Xen, an open source paravirtualized virtual machine monitor (hypervisor), has been adopted by many leading data centers of the world today. A scheduler in Xen handles CPU resources sharing among virtual machines hosted on the same physical system. This study is focused on a scheduler in the current Xen release - the Credit scheduler. Credit uses two parameters (weight and cap) to fine tune CPU resources sharing. Previous studies have shown that these two parameters can impact various performance measures of virtual machines hosted on Xen. In this study, we present a holistic procedure to establish performance models of virtual machines. Empirical data of two commonly used measures, namely calculation power and network throughput, were collected by simulations under various settings of weight and cap. We then employed a powerful machine learning tool (multi-kernel support vector regression) to learn performance models from the empirical data. These models were evaluated satisfactorily by using established procedures in machine learning.**

1. INTRODUCTION

Cloud computing utilizes a pool of virtualized computer resources to offer enterprises on-demand computing facilities. Many Fortune 500 companies are already using the technology or actively researching it. Virtualized computer resources can range from a high level software resource such as salesforce.com to a low level infrastructure resource such as Amazon's EC2.

It is no doubt that virtualization is a key technology in cloud computing to render on-demand provisioning of virtual services. Infrastructure as a service (IaaS) includes the virtualization of compute nodes, storage nodes and network nodes among other possible virtual computing facilities. Today's commodity PCs are shipped with multi-core CPUs, making the job of hosting several virtual machines (VMs) on a physical system very viable. Each VM may run in a simplified environment to serve a single purpose. For example, a database server and a web server can be provided in two separate VMs. Using VMs, servers are more easily administered and their faults can be isolated. Server consolidation can not only improve service reliability but also increase hardware utilization rate.

It is known that VMs can offer many advantages to a business owner. But, the performance of VMs has not been thoroughly studied. This is partly due to the complexity involved in an environment hosting VMs, and partly due to the lack of methodology to model VM performance. Without a reasonable means to model VM performance, the full power of VMs and related technologies in cloud computing can not be unlocked.

Xen [2], an open source paravirtualized virtual machine monitor (hypervisor), has been adopted by many leading data centers of the world today. As a thin layer atop the host machine, Xen exposes the underlying hardware system to VMs efficiently. Xen manages many resources including CPU, memory and I/O, and presents them to VMs coherently. To manage the CPU resources, Xen has provided several CPU schedulers.

The default scheduler in the current Xen release 3 is the Credit scheduler, which takes two parameters (weight and cap) for each domain to fine tune CPU resources sharing. Previous studies have shown that these two parameters can impact various performance measures of VMs hosted on Xen [4, 12]. We would like to push the understanding of these impacts further in this study.

This study is organized as follows. We first discuss related work in the next section. This is followed by the materials and methods section. Section 4 is devoted to our experiment design such as the test environment and performance simulation procedures. We then use regression tools in machine learning to learn performance models from the simulated data. The results are shown in section 5. We conclude this study with a few remarks in section 6.

DOI: 10.5176_2010-2283_1.1.19

2. RELATED WORK

Scheduling, a key concept in operating systems (OSs) design, has been extended to VM hypervisor. Modern OSs typically run more processes than the number of physical CPUs to run them. An OS scheduler coordinates various computer resources (CPU, memory, I/O, etc.) to support concurrently running processes. Likewise, a VM hypervisor schedules various resources to different VMs running on the same host.

Instead of handling I/O drivers in a hypervisor, the current release of Xen lets a host domain (Dom0) handle these driver requests. Based on this observation, Cherkasova et al [4] posted that the relative CPU resources allocated to Dom0 and Dom1 can have an impact on the performance of Dom1. By varying the relative weights of Dom0 and Dom1, the researchers found that performance of a web server running on Dom1 varied accordingly.

Xu et al. [12] presented a comparative performance evaluation of application server consolidations in different configurations of scheduler parameters. The benchmarks considered include cal (calculation power), netperf (network throughput), iozone (disk), httperf (web) and sysbench (database). By varying the Credit parameters of different VMs, the researchers discovered task performances in VMs varied.

Since Dean and Ghemawat [6] published the seminal MapReduce paper in 2004, the MapReduce framework has become a very important technology utilizing seemingly infinite resources in cloud computing. Many machine learning tools are being converted to a MapReduce version [5], which requires two key computing resources, namely calculation and networking. Therefore, we consider two main goals (calculation and networking) served by VMs running on a multi-core system. With a careful design of the simulation procedure, 'cal' and 'netperf' performance data will be collected. These empirical data will be fit by regressions to form performance models.

3. MATERIALS AND METHODS

3.1. CPU Schedulers of Xen

CPU scheduling for VMs has borrowed ideas from process scheduling in OS research. The first criterion to classify CPU scheduling is the agility level that a scheduler responds to VM's CPU requests. A proportional share (PS) scheduler, under the assumption of an *instantaneous* form of CPU sharing, allocates CPU to VMs in proportion to the number of shares of each VM. If the sharing accounting is based on a coarser time granularity, it becomes a fair share (FS) scheduler [4].

The second criterion to classify CPU schedulers is based on the support of work-conserving (WC-mode) and/or non work-conserving (NWC-mode) modes. In the WC-mode, the shares are merely guarantees, while the shares are caps in the NWC-mode.

The third criterion used to classify CPU schedulers is based on the time point when a scheduler reruns the scheduling decision. A preemptive scheduler re-computes the scheduling decision when a new client becomes ready to run. A non-preemptive scheduler reruns the scheduling decision only when the running client gives up the CPU [4].

The Credit scheduler is a non-preemptive scheduler. It supports the PS mechanism through the weight parameter, and the WC-mode and NWC-mode through the cap parameter [4]. Each domain including the host domain receives a default weight of 256 and a default cap of 0. Legal weights range from 1 to 65535. The cap parameter fixes the maximum amount of CPU a domain can consume. The default setting of 0 means there is no upper cap (WC-mode). A cap of 100 indicates one physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc.

3.2. Support Vector Regressions (SVRs)

A support vector regression minimizes the structural risk composed of empirical error and model complexity. This is translated into a convex quadratic programming problem, which is solved in its Lagrange dual form [10]:

$$\max_{\alpha,\alpha^*} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)Q_{ij} - \varepsilon\sum_{i=1}^{n}(\alpha_i^*+\alpha_i) + \sum_{i=1}^{n} y_i(\alpha_i^*-\alpha_i)$$

$$\sum_{i=1}^{n}(\alpha_i^*-\alpha_i)=0$$

$$0 \le \alpha_i^*, \alpha_i \le C, i=1,\ldots n$$

Here, C is a regularizing parameter trading off empirical error with model complexity; ε is the insensitivity measure used in a loss function, $Q_{ij}=K(\vec{x}_i,\vec{x}_j)$ is a kernel matrix computing the inner product $<\phi(\vec{x}_i),\phi(\vec{x}_j)>$, and $\phi(\vec{x}_i)$ is a mapping from the input space to a feature space. Solving the dual problem gives a regression function as

$$f(\vec{x})=\sum_{i=1}^{n}(\alpha_i^*-\alpha_i)K(\vec{x}_i,\vec{x})+b \quad (1)$$

3.3. Multi-kernel SVRs (mkSVRs)

To better cope with varying data density (e.g., data with a mixture distribution), Lanckriet et al. [8] proposed a semi-definite programming method to learn the kernel matrix automatically. Instead of a single kernel, the researchers considered multiple kernels K_j's and their nonnegative linear combinations:

$$K=\sum_{j=1}^{m}\mu_j K_j, \mu_j \ge 0, trace(K)=\rho$$

The objective of mkSVR is to find the best weight for each kernel so that the structural risk is as small as possible. Using a dual representation, this is expressed as a quadratically constrained quadratic programming (QCQP) problem [9]:

$$\max_{t,\alpha^*,\alpha} 2\sum_{i=1}^{n} y_i(\alpha_i^* - \alpha_i) - 2\varepsilon\sum_{i=1}^{n}(\alpha_i^* + \alpha_i) - \rho t$$

$$t \geq \frac{1}{trace(K_j)} \sum_{i,k=1}^{n}(\alpha_i^* - \alpha_i)(\alpha_k^* - \alpha_k)K_j(\vec{x}_i, \vec{x}_k),$$

$$j = 1,..,m$$

$$\sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0,$$

$$0 \leq \alpha_i^*, \alpha_i \leq C, i = 1,...,n$$

The optimal weights $\hat{\mu}_j, j = 1,...,m$ can be found from the dual variables of the optimal solutions, and the regression function is given by

$$f(\vec{x}) = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\sum_{j=1}^{m}\hat{\mu}_j K_j(\vec{x}_i, \vec{x}) + b \qquad (2)$$

4. EXPERIMENT DESIGN

In order to build performance models for VMs hosted on Xen, we consider a simulation environment similar to the intended use of VMs. Since our cloud environment is focused on MapReduce applications, we have selected calculation and networking to model VMs performance.

Our test bed is a commodity computer with an Intel Pentium core 2 CPU (E2180); each core runs at the speed of 2.0 GHz, the RAM size is 2 GB, and the hard disk is an IDE drive with 160 GB. The host OS is CentOS 5.4, which runs Xen 3.1.2. Two Fedora Core 11 guest domains have been created with a RAM size of 256 MB each. One guest domain (VM1) runs calculation intensive tasks, and the second guest domain (VM2) runs network intensive tasks. Benchmark tests are run simultaneously on both VMs.

For the network intensive domain (VM2), we have selected the standard netperf package[1] to measure network throughput. This VM continuously sends network packets to a second physical host in the same local area network for netperf to collect the throughput data. We limit a test period of 10 seconds for each netperf test. Our network backbone is a 100 Mb/sec local area network, and netperf reports a typical test result of tens of Mb/sec. The actual result depends on the Credit parameters set for this VM.

For the calculation intensive domain (VM1), originally we selected the industry standard benchmark packages like unixbench[2] and BYTEmark[3]. These packages can compute CPU benchmarks like Dhrystone and Whetstone for integer and floating point operations. Unfortunately, these packages are too smart to automatically adjust themselves for varying cap parameter to get a stable report. The end result is we can not see the impact of cap on VM1. This has forced us to write our own calculation benchmark test.

We decided to use the Timer and TimerTask classes in Java to determine how many floating point operations can be performed within a fixed period, say 10 seconds, in VM1. The floating point operation is Math.sin(Math.random()). The Java Timer class faithfully observes the time clock in VM1. When the fixed period is set to 10 seconds, regardless of the cap value, the Java program terminates in 10 seconds and reports how many floating point operations have been performed. The benchmark result depends on the cap and weight values set for VM1. For a 10-second run, this is roughly 1 to 8 Mops/sec.

In order to consider the impact of Credit parameters on VMs performance, we have considered several test cases. For VM1, we restricted the weight parameter to the values of 256, 512, 768 and 1024. The cap parameter of VM1 was restricted to the values of 25, 50, 75 and 100. VM2 had the same ranges of weight and cap as VM1. For Dom0, we considered only the weight parameter with the same range as VM1. There was no CPU cap (cap=0) for Dom0. Overall, there were 1024 combinations of weight and cap parameters for two VMs and Dom0.

Each weight and cap parameter combination was used to measure the calculation power ('cal') of VM1 and the network throughput ('netperf') of VM2. In order to minimize the performance fluctuation among different runs of benchmark tests, each 'cal' and 'netperf' test was run five times with a 10-second run period each. The 'cal' and 'netperf' performance result was the average value from these five runs. Linux shell scripts have been written to perform these simulations automatically. At the beginning of a minute, Dom0 sets the Credit parameters for each domain. The shell program sleeps 5 seconds for the new setting to become effective in each domain. Then, the shell program on Dom0 starts 5 runs of 'cal' test in VM1 and 5 runs of 'netperf' test in VM2 at the same time. A single simulation run for a parameter combination takes 1 minute to finish. The total simulation time was 1024 minutes.

5. EXPERIMENTAL RESULTS

We have collected 1024 simulation data. Each data consists of a parameters setting (c1, w1, c2, w2, w0), and two performance measures ('cal', 'netperf'). Here c1 and c2 are cap values for VM1 and VM2 respectively, and w1, w2 and w0 are weight values for VM1, VM2 and Dom0. Thus, we have five independent variables and two dependent variables. Our next step is to model these two dependent variables by the five independent variables via different regression techniques.

We considered three regression techniques in this study: multiple linear regression (MLR), support vector regressions and multi-kernel SVR. For SVRs, we assumed a kernel type of radial basis function (RBF) $K(\vec{x}, \vec{y}) = \exp(-\gamma |\vec{x} - \vec{y}|^2)$. Two γ values (10 and 1) have been used in the study. The SVR with γ =10 is denoted as SVR1 and the other denoted as SVR2. For all SVR based algorithms in this study, we considered an ε = .01 insensitivity loss function. The regularizing parameter

[1] www.netperf.org
[2] www.tux.org/pub/tux/benchmarks/System/unixbench
[3] www.tux.org/~mayer/linux/bmark.html

C was 1 for the 'cal' measure and 100 for the 'netperf' measure. For mkSVR, we have considered five RBFs with γ equal to 100, 10, 1, .1 and .01. Optimal weights $\bar{\mu}_1, \bar{\mu}_2, \bar{\mu}_3, \bar{\mu}_4, \bar{\mu}_5$ will be determined from a QCQP problem in mkSVR.

In order to minimize the variation of independent variables, all five independent variables have been scaled to be between 0 and 1. For the MLR technique, we used Weka [7] to find the linear combination coefficients; for the SVRs, we used libSVM [3]; and for the QCQP problem in mkSVR, we used Mosek [1].

5.1. Out-of-Sample Tests

Our first assessment of the performance modeling is concerned with out-of-sample tests. Frequently in machine learning [11], a data set is split into a training set and a test set. The training and test sets do not overlap. One then applies a learning algorithm (MLR, SVR, mkSVR) to the training set, and assesses the regression model on the test set. We used two indices to measure the performance of a regression model. Let $(\vec{x}_i, t_i), i = 1,...,N$, be a data point of the test set. Here $\vec{x}_i$ is a vector of independent variables and t_i is the target value. Assume y_i is the predicted value for $\vec{x}_i$ from a regression model. The root mean square error (RMSE) and correlation (Corr) value are defined as

$$RMSE = \sqrt{\sum_{i=1}^{N}(t_i - y_i)^2 \Big/ N}$$

$$Corr = \sum_{i=1}^{N}(t_i - \bar{t})(y_i - \bar{y}) \Big/ \sqrt{\sum_{i=1}^{N}(t_i - \bar{t})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Here $\bar{t}$ and $\bar{y}$ are respectively the average of target and predicted values. Based on these definitions, the smaller (larger) the RMSE (Corr), the better a learning algorithm has performed. The data set was randomly split into a training set (~80%) and a test set (~20%). Ten random partitions have been created with the actual size of the training set enclosed by parentheses after a run number in Table 1. Using the 'cal' measure, the out-of-sample test results are reported in Tables 1 and 2.

The first run has 804 training samples and 220 test samples. RMSEs of mkSVR, SVR1, SVR2 and MLR are .171, .238, .271 and .456 respectively. The other rows are interpreted similarly. Table 1 shows that mkSVR has the smallest RMSE among all four learning algorithms in each run. mkSVR also has the smallest average and standard deviation from these ten runs.

Applying a pairwise t-test to the data in Table 1, we find that differences in RMSE are significant among four algorithms. The 2-tail p-value for most pairwise t-tests is smaller than .001 except the test between SVR1 and SVR2. With a p-value of .039, the difference of RMSE between SVR1 and SVR2 is still significant.

Table 1. RMSE for the 'cal' model.

Run #	mkSVR	SVR1	SVR2	MLR
1(804)	.171	.238	.271	.456
2(812)	.195	.290	.272	.444
3(836)	.193	.259	.298	.461
4(792)	.184	.290	.268	.453
5(839)	.170	.274	.261	.421
6(808)	.160	.218	.256	.428
7(817)	.196	.265	.307	.431
8(815)	.174	.244	.286	.450
9(825)	.170	.233	.263	.435
10(818)	.176	.248	.279	.448
Avg	.1789	.2559	.2761	.4427
Stdev	.0124	.0241	.0165	.0133

Table 2 lists the correlation value of target and predicted values for the same out-of-sample test runs in Table 1. Again, mkSVR has predicted values most correlated to the target values among all four learning algorithms. P-values of pairwise t-tests show that the improvement of mkSVR over SVR1, SVR2 and MLR is significant. The 'netperf' out-of-sample tests were observed similarly.

Table 2. Correlation for the 'cal' model.

Run #	mkSVR	SVR1	SVR2	MLR
1(804)	.997	.996	.993	.980
2(812)	.997	.995	.994	.983
3(836)	.997	.995	.992	.979
4(792)	.997	.994	.993	.980
5(839)	.997	.995	.994	.983
6(808)	.998	.997	.994	.984
7(817)	.997	.995	.991	.982
8(815)	.997	.996	.993	.982
9(825)	.998	.997	.994	.983
10(818)	.997	.996	.993	.982
Avg	.9972	.9956	.9931	.9818
Stdev	.0004	.0010	.0010	.0016

5.2. In-Sample Tests and Performance Models

After the out-of-sample tests, we conducted an in-sample test to compare all four learning algorithms. The in-sample test uses the whole data set to train a model and the same set (in-sample) to assess the model. Table 3 shows the results of in-sample tests for the 'cal' model. Again, mkSVR has an RMSE several orders smaller than that of the other three algorithms. The optimal $\bar{\mu}$ weights in equation (2) are listed in Table 4. One can see that RBF kernels $\exp(-.1|\vec{x} - \vec{y}|^2)$ and $\exp(-.01|\vec{x} - \vec{y}|^2)$ did not contribute at all since their weights were zero. Using these optimal weights, the Lagrange multipliers and equation (2), we built a 'cal' performance model for VM1. This model could estimate the calculation power of VM1, given Credit parameters c1, w1, c2, w2 and w0.

Table 3. In-sample tests for the 'cal' model.

	mkSVR	SVR1	SVR2	MLR
RMSE	.0098	.1054	.2503	.4359
Corr	1.0000	.9993	.9943	.9825

Table 4. Weights for mkSVR in 'cal' model.

$\hat{\mu}_1$	$\hat{\mu}_2$	$\hat{\mu}_3$	$\hat{\mu}_4$	$\hat{\mu}_5$
.905	1.693	2.402	0	0

The in-sample assessment on 'netperf" measure is reported in Table 5. Again, mkSVR has provided the best results. Optimal weights of mkSVR for the 'netperf" model are reported in Table 6. A network performance model of VM2 was built from equation (2) using these parameters.

Table 5. In-sample tests for the 'netperf" model.

	mkSVR	SVR1	SVR2	MLR
RMSE	.0098	1.4431	3.8124	8.9387
Corr	1.0000	.9924	.9530	.6471

Table 6. Weights for mkSVR in 'netperf" model.

$\hat{\mu}_1$	$\hat{\mu}_2$	$\hat{\mu}_3$	$\hat{\mu}_4$	$\hat{\mu}_5$
1.942	2.615	.443	0	0

6. CONCLUSION

Virtual machines hosted on Xen are becoming more popular in cloud computing applications. Using commodity computers and networking equipments, server consolidation is no longer restricted to big enterprises only. Small and medium size businesses can utilize their commodity computers and Xen to consolidate their application servers.

It has been found that parameters of the Credit scheduler in Xen can affect the performance of VMs hosted on Xen [4, 12]. In this study, we push this line of research further by establishing performance models of VMs. We have presented a holistic procedure to build the models. This procedure started with well-designed benchmark simulations to collect empirical performance data. The simulations should be conducted in an environment as close as possible to the intended use of VMs. In this study, we assumed a scenario of two VMs hosted on Xen with one VM running calculation intensive jobs and the other VM running network intensive jobs.

After the empirical data has been collected, we employed regression techniques from machine learning to learn the performance models. We found that mkSVR has provided the best results in terms of RMSE and correlation value, and thus used mkSVR to build our final models.

One simple application of our performance models is in the area of load balancing VMs hosted on a physical host. For example, by knowing the performance models of VM1 and VM2, we can consider a two-objective optimization problem to maximize these two performance functions simultaneously. Credit parameters (c1, w1, c2, w2, w0) maximizing these two functions can be enforced properly in order to get the optimal results from VM1 and VM2 at the same time.

The environment of VMs hosted on Xen is very complicated. In this study, we only considered a scenario of two VMs, and this scenario has created enough demand in simulations to collect empirical data. If there were three VMs, we would have to consider parameter combinations of the type w1, c1, w2, c2, w3, c3, w0. This will no doubt create a much higher demand in simulations to collect empirical data.

Though this study considered only the Credit CPU scheduler in Xen, we believe that the general procedure outlined in the study can be used to include other Xen managed resources such as memory. The procedure includes (1) design proper benchmark simulations to collect empirical data; (2) find performance models from empirical data by regressions.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Economic Affairs (Taiwan) under the grant number 98-EC-17-A-02-S2-0114 and the National Science Council (Taiwan) under the contract number NSC98-2410-H-366-001.

REFERENCES

[1] Andersen, E. and Andersen, A., "The MOSEK interior point optimization for linear programming: an implementation of the homogeneous algorithm", In H. Frenk, C. Roos, T. Terlaky and S. Zhang (editors), *High Performance Optimization*, pp. 197-232, Kluwer Academic Publishers, 2002.

[2] Brahm, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the art of virtualization", In *Proceeding of the 19th ACM symposium on operating systems principles*, October 2003.

[3] Chang, C. and Lin, C., LIBSVM: *a library for support vector machines*, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.

[4] Cherkasova, L., Gupta, D., and Vahdat, A., "Comparisons of the three CPU schedulers in Xen", *Sigmetrics Performance Evaluation Review*, 35, 2, September 2007, pp. 42-51.

[5] Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, G., Ng, A., and Olukotun, K., *Map-Reduce for machine learning on multi-core*, http://www-cs.stanford.edu/~ang/papers/nips06-mapreducemulticore.pdf, 2006.

[6] Dean, J. and Ghemawat, S., "MapReduce: simplified data processing on large clusters", *Communication of the ACM*, 51, 1, January 2008, pp. 107-113.

[7] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I., "The Weka data mining software: an update", *SIGKDD Exploration*, 11, 1, 2009.

[8] Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L., and Jordan, M., "Learning the Kernel Matrix with Semidefinite Programming", *Journal of Machine Learning Research*, 5, 2004, pp. 27-72.

[9] Qui, S. and Lane, T., *Multiple kernel learning for support vector regression*, Computer Science Department, University of New Mexico, Technical Report CS-2005-42, 2005.

[10] Scholkopf, B. and Smola, A., *Learning with Kernels*, MIT Press, 2002.

[11] Witten, I. and Frank, E., *Data Mining, Practical Machine Learning Tools and Techniques*, Morgan Kauffman Publishers, 2005.

[12] Xu, X., Shan, P., Wan, J., and Jiang, Y., "Performance evaluation of the CPU scheduler in Xen", In *Proceeding of International Symposium On Information Science and Engineering*, 2008, pp. 68-72.

Dr. Doong received his Ph.D. in Mathematics from the University of California at Berkeley. He is an Associate Professor at ShuTe University in Taiwan.

Dr. Lai received his Ph.D. in Information Engineering from the National Central University in Taiwan. He is an Associate Professor at National Kaohsiung University in Taiwan.

Dr. Lee received his Ph.D. in Computer Science from the University of North Carolina at Chapel Hill. He is a Professor at the National Sun Yat-Sen University in Taiwan.

Dr. Ouyang received his Ph.D. in Electric Engineering from the National Sun Yat-Sen University in Taiwan. He is an Assistant Professor at I-Shou University in Taiwan.

Dr. Wu received his Ph.D. in Electric Engineering from the National Sun Yat-Sen University in Taiwan. He is an Associate Professor at National Kaohsiung University in Taiwan.